

Client/Server Security

Implementing SQL Security Using Application Roles

By *Charles L. Collins*

Client/server database applications must frequently address the need to enforce access and usage constraints for various classes of users. We are all familiar with the Human Resources example where everyone can view (SELECT) the basic columns of the Employee table, but only managers can view (SELECT) the Salary column, and only HR administrators can change (UPDATE) the Salary column.

We're also familiar with the basic mechanisms (views and privileges) that Oracle provides to handle these situations from the database perspective. As we'll see, however, these basic mechanisms have their limitations. Moreover, from an application perspective, there is no standard mechanism by which we can selectively hide the Salary field on the "Employee Search" form of our HR application based on who the current user is, or eliminate access to entire chunks of application functionality (dealing with salary administration, for instance). This article presents the design of a security system that uses Oracle's database roles to address both these issues and to simplify effective security handling in client/server applications.

Strategies

Discussion of what security strategy to adopt usually focuses on a choice between a *front-end* (client-based) strategy and a *back-end* (server-based) strategy, according to which "end" is primarily responsible for enforcing security. The choice is not really binary, however, as there are a variety of possible strategies, and all involve both ends to some degree.

The "extreme" front-end strategy grants database privileges to all application users (or to PUBLIC) on all objects used by the application at the least restrictive level used anywhere by the application. That is to say, if the application *ever* needs to update a certain table, then *all* application users are granted UPDATE privileges on that table. If, in fact, only certain users are eligible to update the table, or if it's appropriate to update the table only under certain circumstances, it's up to the application to enforce these more restrictive conditions.

Apart from the fact that this strategy inevitably increases application complexity, it also leaves open

the possibility that any privileged user might gain access to the database objects in question via some means (Access, Paradox, SQL*Plus, etc.) other than the controlled application, and thereby, in the absence of appropriate application enforcement, do something undesirable to the database, whether intentionally or not.

All other strategies give more responsibility to the database. One approach is to identify the specific access requirements for each application user and to grant appropriate database object privileges to individual users on that basis. This still leaves open the possibility that a user privileged to perform a certain operation under application control might do something undesirable through some other means.

Role Playing

A generalization of the user-based approach takes advantage of Oracle *roles*. Distinct roles are created for various classes of users (e.g. managers, HR administrators), and database object privileges are granted to the roles rather than to the individual users. From a security point of view, this approach suffers from the same defects as the user-based approach. It's just more convenient to administer.

Oracle provides the further capability, however, to selectively enable or disable a role, as well as to enable by default any or all of the roles assigned to a user. Roles assigned to classes of users as described above must necessarily be enabled by default in order to be useful to all applications. This opens up the same old security hole.

An alternative is to create an *application role* for each application that uses the database. Such a role is not enabled for any user by default, but is specifically

enabled at run time by the application that uses it. Just those database object privileges appropriate to the application are granted to the application role, and just those users — or classes of users, in the form of other roles — authorized to use the application are granted the application role. Since the role is enabled only during the execution lifetime of the application, we avoid the problem of inappropriate uncontrolled access.

(Or “innocent” inappropriate access, at least. Nothing about application roles as such — even those with passwords — prevents a determined attacker with some legitimate database privileges in the context of application A from writing his own application B that masquerades as A and then attempts to misuse the privileges granted to A.)

Yet application roles take only one step in the direction we want to go. In the case where not every application user is authorized to do everything the application can do, we would have to define distinct application roles for the different classes of application users, grant the appropriate privileges to each role, and have the application enable the appropriate role as needed.

This addresses the security issues, but may be difficult to set up and administer. And integrating application behavior with database privileges and controlling access to functionality within the application appropriately are still left to be dealt with in an ad hoc manner, providing ample opportunity for error.

Make It Clear

What we need to do is make explicit the relationships between user identity, application functionality, and database object privileges. Not only can we use this information to customize application behavior at run time, but we can also use it as the basis for creating all the user/usage-specific database roles we need — completely automatically. Let’s see how this might be done.

What follows is a design for a system that facilitates defining privileges for application functionality and database objects at as fine a level of granularity as may be required, and that allows applications to make use of this information at run time with a minimum of coding effort.

Specifically, the system:

- makes explicit the notion of a usage “right,” or privilege, for any application element (e.g. form, button, method, menu item; anything an application needs to control).
- provides a mechanism to associate database object privileges (INSERT, SELECT, UPDATE, UPDATE(column), DELETE, EXECUTE on any table/view/procedure) with application privileges in a single security database.
- provides a mechanism whereby an application can easily determine at run time what privileges are available to the current user, and dynamically activate just those database object privileges appropriate to the application context at hand and the current user’s application privilege in that context.

The system consists of an administration application and a run-time component to be imbedded in, or loaded by, applications using secu-

rity services. The security system data is itself stored in the form of database tables. These are summarized in [Figure 1](#) and described in detail below. An example follows.

The fundamental context for all privileges is the Application, identified by its name. Associated with each Application may be any number of User Roles, also simply identified by name. The User_Role table associates application users with User Roles as required.

The App_Privilege table associates with each Application/User Role combination any number of Application Privileges assigned to particular Application Items. Privileges too are simply names, invented at the discretion of the designer. They might include Create, Insert, Read, Select, Update, Change, Delete, Execute, View, Manage, etc., but are not limited to these or to any fixed set. An Application Item is any notional application element (form, menu, menu item, sub-system) whose availability and/or usage an application wishes to control.

The DB_Privilege table associates with each role/item/privilege combination, any number of Database Privileges assigned to particular Database Objects. A Database Privilege is any privilege recognized by the DBMS the application uses, and a Database Object is any object on which the DBMS recognizes privileges.

From this information, as formulated by the application designer, the security administration application automatically synthesizes database object privilege groups (Oracle roles), records their (generated) names in the security tables, and creates (or updates) them in the DBMS.

For example (see [Figure 2](#)), consider a highly simplified human resources application. The application defines three user roles: Administrator, who can do everything; Manager, who can view all information, but change nothing; and Viewer, who can only view the organization’s management structure.

Named privileges on designated application elements are assigned to the user roles as in the App_Privilege table (see [Figure 3](#)). Remember that the Database Role information is not entered by the security system user, but is generated automatically by the system.

The database object privileges required to support the various application privileges in the designated application contexts are specified in the DB_Privilege table (see [Figure 4](#)). With this information in hand, the security system defines roles and grants privileges as in [Figure 5](#). If entering the information into the tables in [Figure 2](#) through [Figure 4](#) seems tedious, consider that it saves devising, entering, and maintaining what appears in [Figure 5](#). And this is for a highly simplified example.

On application startup, the security run-time component initializes itself by retrieving from the security tables the information necessary to populate a private data structure that associates each application item for the application at hand (as defined at design time) with the present user’s application privilege and, if applicable, database role for that item. Caching this information locally makes further calls to the security component in the application very fast.

While the application is running, as control passes into each designated application context, the application, without concern for any of the underlying information, including the name of the role, can request the security system to enable (SET ROLE) the appropri-

Table Name	Key				Attributes	
User_Role	User ID	Application Name			User Role	Database Role
App_Privilege	Application Name	User Role	Application Item		Application Privilege	Database Role
DB_Privilege	Application Name	Application Item	Application Privilege	Database Object	Database Privilege	

Figure 1: Summarizing the security system.

Table: User_Role

User ID	Application Name	User Role	Database Role
Laszlo	HR Admin	Administrator	APP001
Fumiko	HR Admin	Manager	APP002
Jiri	HR Admin	Viewer	APP003
Marthe	HR Admin	Viewer	APP003

Table: App_Privilege

Application Name	User Role	Application Item	Application Privilege	Database Role
HR Admin	Administrator	Salary Guidelines		
HR Admin	Administrator	Salary Grid	Modify	OBJ001
HR Admin	Administrator	Employee Summary	Modify	OBJ002
HR Admin	Administrator	Organization	Modify	OBJ003
HR Admin	Manager	Employee Summary	View Salary	OBJ004
HR Admin	Manager	Organization	View	OBJ005
HR Admin	Viewer	Employee Summary	View	OBJ006
HR Admin	Viewer	Organization	View	OBJ005

Table: DB_Privilege

Application Name	Application Item	Application Privilege	Database Object	Database Privilege
HR Admin	Salary Grid	Modify	SALARY	MODIFY
HR Admin	Employee Summary	Modify	EMPLOYEE	MODIFY
HR Admin	Employee Summary	Modify	SALARY	SELECT
HR Admin	Organization	Modify	EMPLOYEE	SELECT
HR Admin	Organization	Modify	EMP_DEPT	MODIFY
HR Admin	Organization	Modify	DEPARTMENT	MODIFY
HR Admin	Employee Summary	View Salary	EMP_MGRVIEW	SELECT
HR Admin	Employee Summary	View	EMP_EMPVIEW	SELECT
HR Admin	Organization	View	EMPLOYEE	SELECT
HR Admin	Organization	View	EMP_DEPT	SELECT
HR Admin	Organization	View	DEPARTMENT	SELECT

Figure 2 (Top): The User_Role table.

Figure 3 (Middle): The App_Privilege table.

Figure 4 (Bottom): The DB_Privilege table. To reduce the size of the example, the privilege name MODIFY is used in place of the three distinct privileges INSERT, UPDATE, and DELETE.

ate database role for the context at hand. In this way, the user receives appropriate privileges on just those database objects relevant to the context at hand, and just for the time during which access is required (provided the application notifies the security component when control passes out of the context in question).

To adjust its own behavior (to disable a button or menu item, for example), the application can ask the security system to supply (as a string or list of strings) the user's application privilege(s) in the current context, on the basis of which the application can take appropriate action. In a GUI application, database roles can easily be made to track the active window (form) by putting the appropriate security system calls — `enableRole(context)`, `disableRole()` — in the `FormActivate` and `FormDeactivate` events, respectively.

```
CREATE ROLE app001; /* Administrator */
GRANT app001 TO laszlo;
```

```
CREATE ROLE obj001; /* Salary Grid, Modify */
GRANT INSERT, UPDATE, DELETE ON salary TO obj001;
GRANT obj001 TO app001;
```

```
CREATE ROLE obj002; /* Employee Summary, Modify */
GRANT INSERT, UPDATE, DELETE ON employee TO obj002;
GRANT SELECT ON salary TO obj002;
GRANT obj002 TO app001;
```

```
CREATE ROLE obj003; /* Organization, Modify */
GRANT SELECT ON employee TO obj003;
GRANT INSERT, UPDATE, DELETE ON emp_dept TO obj003;
GRANT INSERT, UPDATE, DELETE ON department TO obj003;
GRANT obj003 TO app001;
```

```
CREATE ROLE app002; /* Manager */
GRANT app002 TO fumiko;
```

```
CREATE ROLE obj004; /* Employee Summary, View Salary */
GRANT SELECT ON emp_mgrview TO obj004;
GRANT obj004 TO app002;
```

```
CREATE ROLE obj005; /* Organization, View */
GRANT SELECT ON employee TO obj005;
GRANT SELECT ON emp_dept TO obj005;
GRANT SELECT ON department TO obj005;
GRANT obj005 TO app002;
```

```
CREATE ROLE app003 /* Viewer */
GRANT app003 TO jiri;
GRANT app003 TO marthe;
```


```
CREATE ROLE obj006; /* Employee Summary, View */
GRANT SELECT ON emp_empview TO obj006;
GRANT obj006 TO app003;
```

Figure 5: Defining roles and granting privileges.

Conclusion

Thanks to Oracle's flexible and powerful roles, the answer to the question of which system component should be in charge of client/server security turns out to be neither the client nor the server, but the security itself, that is, the security information.

By explicitly and systematically taking account of the relationships between application objects and database objects, synthesizing this information in the form of roles and then using the same security database to control both client and server, we achieve two important benefits:

- We facilitate the development, testing, and maintenance of any client/server application by reducing and simplifying application code and by completely eliminating the need to write any security-related database code.
- We improve the overall software development process — facilitating development and testing of all client/server applications and improving predictability and repeatability — by providing a common framework and reusable components. 

Charles Collins is a computer scientist at PCSI, Inc., specializing in very large relational databases and multi-dimensional access to them. He can be reached via CompuServe (102336,2252), or at PCSI at (201) 816-8002.



Available December 1996

The Must Have Reference Source For The Serious Oracle® Developer



A \$130 Value
Available Now for only

\$49.95

California residents
add 7½% Sales Tax,
plus \$5 shipping & handling
for US orders.
(International orders add \$15 shipping & handling)

The Entire Text of all Technical
Articles Appearing in
Oracle® Informant® in 1996

The Oracle® Informant® Works 1996 CD-ROM
will include:

- All Technical Articles
- Text and Keyword Search Capability
- Improved Speed and Performance
- All Supporting Code and Sample Files
- 16-Page Color Booklet
- Third-Party Add-In Product Demos
- CompuServe Starter Kit with \$15 Usage Credit.

Call Now Toll Free
1-800-88-INFORM

1-800-884-6367 Ask for offer # WEB
To order by mail,
send check or Money Order to:
Informant Communications Group, Inc.
ATTN: Works CD offer # WEB
10519 E. Stockton Blvd, Suite 142
Elk Grove, CA 95624-9704
or Fax your order to 916-686-8497

Order Now!

To order, mail or fax
the adjoining form or call
(916) 686-6610 Fax: (916) 686-8497

International rates

Magazine-only

\$54.95/year to Canada
\$74.95/year to Mexico
\$79.95/year to all other countries

Magazine AND Companion Disk Subscriptions

\$124.95/year to Canada
\$154.95/year to Mexico
\$179.95 to all other countries

California Residents add 7½%
sales tax on disk subscription

Get Informed!

Subscribe to Oracle Informant,
The Independent Monthly Guide to Oracle
Development.

Order Now and Get One Issue FREE!

For a limited time you can receive the first issue FREE plus 12 additional
issues for only \$49.95 That's nearly 25% off the yearly cover price!



Each big issue of *Oracle
Informant* is packed with Oracle
tips, techniques, news, and more!

- Client/Server Development
- Using Developer/2000™
- Tuning Oracle7
- PL/SQL Techniques
- Advanced Oracle Topics
- Distributed Managment
- Product Reviews
- Book Reviews
- News from the Oracle
Community
- Oracle User Group Information

YES!, I want to sharpen my Oracle skills. I've checked the subscription plan I'm interested in below

☐ **Magazine-Only Subscription Plan...**

13 Issues Including One Bonus Issue at \$49.95.

☐ **Magazine AND Companion Disk Subscription Plan...**

13 Issues and Disks Including One Bonus Issue and Disk at \$119.95
The Oracle Informant Companion Disk contains source code, support files, examples, utilities, samples, and more!

☐ **Oracle Informant Works 1996 CD-ROM \$49.95 (Available December 1996)**

US residents add \$5 shipping and handling. International customers add \$15 shipping and handling.

Name

Company

Address

City State Zip Code

Country Phone

FAX E-Mail

Payment Method...

☐ Check (payable to Informant Communications Group) ☐ Purchase Order-- Provide Number

☐ Visa ☐ Mastercard ☐ American Express Card Number

Expiration Date Signature

WEB

Oracle and its products are trademarks of Oracle Corporation